A FORTRAN AUTOPROGRAM FOR SOLVING ORDINARY LINEAR DIFFERENTIAL EQUATIONS WITH CONSTANT COEFFICIENTS USING EXACT Z-TRANSFORMS OF $(1/s)^n$ AND TRAPEZOIDAL CONVOLUTION

by

WILLIAM A. PICKER

B. S., Kansas State University, 1964

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1966

Approved by:

Charles A. Halijak
Major Professor

TABLE OF CONTENTS

INTRODUCTION

Numerical transform techniques were introduced into electrical engineering by Tustin in 1947. Since then, several investigators have produced various methods whose aim is digital solution of ordinary linear and nonlinear differential equations. It has been demonstrated by Halijak (2) that all of these methods have trapezoidal convolution as their fundamental basis.

An appraisal of these methods has led to choosing the multiple integrator substitution program, together with exact z-transforms of $(1/s)^n$, as the preferred method for solving ordinary differential equations with constant coefficients. This method is analogous to Naumov's program (5), but its execution is quite different. Explicitly, this method is applicable only to functions of time continuous on $[0,\infty]$. However, no generality is lost since the method is inherently adaptable to discontinuous functions.

All numerical transform techniques require lengthy algebraic calculations for determining coefficients of recurrence relationships, and these calculations, external to the digital computer, are a large source of human error. The purpose of this thesis is to program a digital computer to execute these lengthy calculations at a high speed, and without human error. The program determines the required recurrence relation, and then iterates this recurrence relation to obtain a solution. Furthermore, since the program accepts an n^{th}-order ($2 \leq n \leq 13$) ordinary differential equation with constant coefficients

without additional computer programming, it can properly be called an "autoprogram".

It should be noted that this autoprogram applies only to initial value problems of ordinary differential equations with constant coefficients.

It is presumed that the reader is familiar with z-transforms and trapezoidal convolution (1). A deviation from standard z-transform notation is employed throughout; namely, $z = e^{-Ts}$ is used in place of the standard e^{Ts}. The delay operator, e^{-Ts}, is physically realizable, whereas the predictor, e^{Ts}, has not been found in this physical world.

DEVELOPMENT OF THE SOLUTION OF THE n^{th}-ORDER EQUATION WITH ZERO INITIAL CONDITION

The Standard Form

The following form will be considered the standard form of the equation to be solved.

$$c_1 \frac{d^n y(t)}{dt^n} + c_2 \frac{d^{n-1} y(t)}{dt} + \cdots + c_{n+1} y(t) = x(t) \quad (1)$$

Initial conditions of $y(t)$ and its $(n - 1)$ derivatives are assumed to be zero. Nonzero initial conditions are considered in the next section.

The Solution

The above standard form is now expressed in closed form using the differential operator $D = \frac{d}{dt}$.

$$\left[\sum_{k=0}^{n} c_{k+1} D^{n-k} \right] y(t) = x(t) \quad (2)$$

The Laplace transform of equation (2) is

$$\left[\sum_{k=0}^{n} c_{k+1} s^{n-k} \right] \bar{y} = \bar{x} \quad (3)$$

where the overbar denotes an s-domain function.

Dividing by the highest power of s, which occurs when $k = 0$, the following form results:

$$\left[\sum_{k=0}^{n} c_{k+1}(1/s)^k\right] \bar{y} = (1/s)^n \bar{x} \tag{4}$$

The next step is to take the z-transform of the equation, using trapezoidal convolution (1) to obtain the transforms of products of $(1/s)^i$ and $\bar{y}$. This method requires that terms involving $(1/s)^0$ and $(1/s)^1$ be treated as special cases, since the term involving $(1/s)^0$ involves merely the transform of the s-variable, i.e., no convolution is involved, and the term involving the $(1/s)^1$ term reintroduces the initial value of the unit step function, which has a value of unity in the time domain. All other terms involving $(1/s)^i$, where $i > 1$, require reintroduction of zero initial values. With this in mind, the following steps result.

$$Z\left(\sum_{k=0}^{n}\left[c_{k+1}(1/s)^k \bar{y}\right]\right) = Z\left[(1/s)^n \bar{x}\right] \tag{5}$$

$$\sum_{k=0}^{n} c_{k+1} Z\left[(1/s)^k \bar{y}\right] = Z\left[(1/s)^n \bar{x}\right] \tag{6}$$

$$c_1 Z\bar{y} + \frac{c_2 T}{2}\,\frac{1+z}{1-z}\,Z\bar{y} + \sum_{k=2}^{n} \frac{c_{k+1}T^k}{(k-1)!}\,\frac{z\,A_k(z)}{(1-z)k}\,Z\bar{y}$$

$$= \frac{T^n}{(n-1)!}\,\frac{z\,A_n(z)}{(1-z)^n}\left[Z\bar{x} - \frac{x_0}{2}\right] \tag{7}$$

The definitions listed below will introduce the reader to new symbols.

T denotes the sampling time interval.

$Z\bar{y}$ denotes the z-transform of $\bar{y}$.

$A_i(z)$ stems from $Z(1/s)^i = \dfrac{T^{i-1}}{(i-1)!} \dfrac{z\, A_i(z)}{(1-z)^i}$ and is derived in reference (3).

The next step toward developing a recurrence relation in terms of z is to clear fractions in the entire equation by multiplying by $(1-z)^n$, and this yields

$$c_1(1-z)^n\, Z\overline{y} + \frac{c_2 T}{2}(1+z)(1-z)^{n-1}\, Z\overline{y} + \sum_{k=2}^{n} \frac{c_{k+1} T^k}{(k-1)!}\, z\, A_k(z)\, (1-z)^{n-k}\, Z\overline{y} = \frac{T^n}{(n-1)!}\, z\, A_n(z) \left[Z\overline{x} - \frac{x_0}{2} \right] \tag{8}$$

This equation is rewritten for simplicity as

$$\left[A(n,z) + B(n,z) + C(n,k,z) \right] Z\overline{y} = D(n,z) \left[Z\overline{x} - \frac{x_0}{2} \right] \tag{9}$$

with the following identifications applying to equation (9):

$$A(n,z) = c_1(1-z)^n \tag{10}$$

$$B(n,z) = \frac{c_2 T}{2}(1+z)(1-z)^{n-1} \tag{11}$$

$$C(n,k,z) = \sum_{k=2}^{n} \frac{c_{k+1} T^k}{(k-1)!}\, z\, A_k(z)\ \ (1-z)^{n-k} \tag{12}$$

$$D(n,z) = \frac{T^n}{(n-1)!}\, z\, A_n(z) \tag{13}$$

Upon examining these coefficients it becomes apparent that both $A(n,z)$ and $B(n,z)$ represent polynomials in z of degree n.

The $C(n,k,z)$ coefficients are polynomials in z for every value of k between 2 and n. To determine the degree of these

polynomials, it should first be noted (3) that $A_i(z)$ is always of degree (i-2). If this is multiplied by $z(1-z)^{n-k}$, the product will always be of degree (n-1), and the z^0 term is absent.

A Closed Form for $z\,A_k(z)(1-z)^{n-k}$

Since the C(n,k,z) coefficient includes a number of polynomials it is of benefit to develop a closed form which would generate any coefficient of any of these polynomials.

The polynomial E(n,k,z), $n \geq 2$ is defined as

$$E(n,k,z) = z\,A_k(z)(1-z)^{n-k} = \sum_{r=1}^{n-1} b_r z^r \tag{14}$$

The problem is now to determine the b_r coefficients. Criswell (3) has shown that

$$A_k(z) = \sum_{q=0}^{n-2} A(k,q)\,z^q \tag{15}$$

Furthermore, the binomial expansion yields

$$z(1-z)^{n-k} = \sum_{q=0}^{n-k} \binom{n-k}{q} (-1)^q\, z^{q+1} \tag{16}$$

The product of equations (15) and (16) is found by employing the "Cauchy Product" (4). The Cauchy product of two polynomials $\sum_{i=0}^{N} c_i x^i$ and $\sum_{j=0}^{M} d_j x^j$ is $\sum_{n=0}^{N+M} e_n x^n$, where $e_n = \sum_{k=0}^{n} c_k d_{n-k}$. Using this relationship, the expression for b_r, for given n and k, becomes

$$b_r(n,k) = \sum_{u=0}^{r} A(k,u) \binom{n-k}{r-u} (-1)^{r-u} \tag{17}$$

Collection of z-Coefficients Associated with $Z\bar{y}$

The task at hand is to collect all coefficients of equal powers of z on the left side of equation (9). This amounts to expanding equations (10), (11), and (12) for a given n, and adding coefficients of like powers of z. This operation is performed by the computer and the illustration below is given to aid in the development of the recurrence relation.

Consider the following polynomials of enforced like degree.

$$
\begin{aligned}
A(n,z) &= c_1\left[1 + a_1 z + a_2 z^2 + \quad \ldots \quad + a_n z^n\right] \\
B(n,z) &= \frac{c_2 T}{2}\left[1 + b_1 z + b_3 z^2 + \quad \ldots \quad + b_n z^n\right] \\
C(n,2,z) &= \frac{c_3 T^2}{1!}\left[0 + d_1 z + d_2 z^2 + \ldots + d_{n-1} z^{n-1} + 0 z^n\right] \\
k=2(1)n \quad &\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
C(n,n,z) &= \frac{c_{n+1} T^n}{(n-1)!}\left[0 + w_1 z + w_2 z^2 + \ldots + w_{n-1} z^{n-1} + 0 z^n\right]
\end{aligned}
\tag{18}
$$

Upon summing coefficients of all columns in this array, there results a single polynomial in z whose degree is n. Thus the left side of equation (9) has been reduced to the form

$$
\left[A_1 + A_2 z + A_3 z^2 + A_4 z^3 + \ldots + A_{n+1} z^n\right] Z\bar{y} = \frac{T^n}{(n-1)!}\, z\, A_n(z)\left[Z\bar{x} - \frac{x_0}{2}\right] \tag{19}
$$

The next task is to reduce the right side of equation (19). This polynomial consists of a constant multiplying two polynomials in z. The first, $z\,A_n(z)$, is of degree (n-1), whereas the forcing function factor $\left[Z\bar{x} - \frac{x_0}{2}\right]$ is a polynomial in z of indefinite degree. The entire term on the right side yields the form

$$= \frac{T^n}{(n-1)!}\left[u_0 + u_1 z + u_2 z^2 + \ldots + u_{n-1} z^{n-1}\right]$$

$$\left[\frac{x_0}{2} + x_1 z + x_2 z^2 + \ldots + x_k z^k + \ldots\right] \qquad (20)$$

where x_0 = initial value of the forcing function

$x_n = x(nT)$, i.e., the forcing function evaluated at $t = nT$.

The resultant polynomial, also of indefinite degree, is written as:

$$= \frac{T^n}{(n-1)!}\left[w_0 + w_1 z + w_2 z^2 + \ldots + w_k z^k + \ldots\right] \qquad (21)$$

The required coefficients in equation (21) are found to be:

$$w_0 = u_0 \frac{x_0}{2}$$

$$w_1 = u_0 x_1 + u_1 \frac{x_0}{2}$$

$$w_2 = u_0 x_2 + u_1 x_1 + u_2 \frac{x_0}{2}$$

$$\vdots$$

$$w_{n-1} = u_0 x_{n-1} + u_1 x_{n-2} + \ldots + u_{n-1} \frac{x_0}{2}$$

$$w_n = u_0 x_n + u_1 x_{n-1} + \cdots + u_{n-1} x_1$$

$$\vdots$$

$$w_k = u_0 x_k + u_1 x_{k-1} + \cdots + u_{n-1} x_{k+1-n}$$

At this point equation (9) has been reduced to the form

$$\left[P(z)\right] Z\bar{y} = Q(z) \tag{22}$$

where $P(z) = \sum_{k=0}^{n} A_{k+1} z^k$

$$Q(z) = \frac{T^n}{(n-1)!} \sum_{k=0}^{\infty} w_k z^k$$

Recurrence Relation

The recurrence relation for the iterated solution is only a few steps away. It remains to translate equation (22) from the z-domain into the time domain. Since $z = e^{-Ts}$ is a delay operator such that every z delays its coefficient by T seconds, and every z^n delays its coefficient by (nT) seconds, the following transitions are employed.

z-domain		Time domain
$Z\bar{y}$	⟶	y_n
$z\ Z\bar{y}$	⟶	y_{n-1}
$z^k Z\bar{y}$	⟶	y_{n-k}

Upon applying these transitions to equation (22), Q(z) emerges as a sequence $\{w_k\}$ in the time domain, where the k^{th} term in the sequence is the coefficient of z^k in the z-domain; hence equation (22) transforms into

$$A_1 y_k + A_2 y_{k-1} + A_3 y_{k-2} + \cdots + A_{n+1} y_{k-n} = \frac{T^n}{(n-1)!} \left[w_k \right] \quad (23)$$

Solving for y_k, one obtains

$$y_k = \frac{1}{A_1} \left[\frac{T^n}{(n-1)!} w_k - A_2 y_{k-1} - A_2 y_{k-2} - \cdots - A_{n+1} y_{k-n} \right] \quad (24)$$

Equation (24) is the final recurrence relation for the iterated solution of the differential equation. For the first n iterations of this equation it is understood that a negative subscript results in a zero value. This equation shows clearly the dependence of y_k on n previous y_k's, and n previous x_k's which appear in w_k. The subscript k ranges from 0 to m, where M depends both on the sampling time T and the desired solution time.

The constant A_1 depends on the coefficients of the differential equation and the sampling time T. Equations (18) show that $A_1 = c_1 + c_2 T/2$; hence if $T = -2c_1/c_2$, A_1 becomes zero, and equation (24) becomes an undefined quantity. This possibility must be investigated when choosing T and a check feature has been incorporated in the program.

This concludes the development of the solution of the n^{th} order differential equation with zero initial conditions.

DEVELOPMENT OF THE SOLUTION OF THE n^{th}-ORDER EQUATION WITH INITIAL CONDITIONS

Equation (1) will again be used as the standard form of the differential equation.

The new problem differs from the previous development only by the presence of the initial values of y(t) and its (n-1) derivatives. Since these values are first introduced during the Laplace transformation, it was found advantageous to employ a method of induction which points out that initial conditions may be considered as an alteration of the forcing function.

The first-order and second-order cases suffice to show the trend, from which a general form is then deduced. The first-order case yields:

$$(c_1D + c_2)y(t) = x(t)$$

$$c_1(s\bar{y} - y_0) + c_2\bar{y} = \bar{x}$$

$$(c_1 + \frac{1}{s} c_2)\bar{y} = \frac{1}{s} \bar{x} + \frac{1}{s} c_1y_0 \tag{25}$$

The second-order case yields:

$$(c_1D^2 + c_2D^1 + c_3)y(t) = x(t)$$

$$c_1(s^2\bar{y} - sy_0 - y_0^{(1)}) + c_2(s\bar{y} - y_0) + c_3\bar{y} = \bar{x}$$

$$(c_1 + \frac{1}{s} c_2 + \frac{1}{s^2} c_3)\bar{y} = \frac{1}{s^2} \bar{x} + y_0(c_1 \frac{1}{s} + c_2 \frac{1}{s^2}) + y_0^{(1)}(c_1 \frac{1}{s^2}) \tag{26}$$

The general form becomes:

$$\left[\sum_{k=0}^{n} \left(\frac{1}{s}\right)^k c_{k+1}\right] \bar{y} = \left(\frac{1}{s}\right)^n \bar{x} + \sum_{k=1}^{n} \left(\frac{1}{s}\right)^k \left[\sum_{i=1}^{k} c_1 y_0^{(k-i)}\right] \quad (27)$$

Upon comparing equations (27) and (4), it becomes apparent that equation (27) differs merely by the term involving the initial conditions. It is also noted that this term adds to the forcing function, and that no convolution is involved.

Development of Recurrence Relation

Development of equation (27) into a recursive relationship follows the procedure of the previous section. It should be pointed out, however, that trapezoidal convolution will reintroduce initial conditions in every convolution operation.

Taking the z-transform of equation (27), we obtain:

$$Z\left[c_1\bar{y}\right] + Z\left[c_2\left(\frac{1}{s}\right)\bar{y}\right] + Z\left[\left(\sum_{k=2}^{n}\left(\frac{1}{s}\right)^k c_{k+1}\right)\bar{y}\right]$$

$$= Z\left[\left(\frac{1}{s}\right)^n \bar{x}\right] + Z\left[c_1 y_0\left(\frac{1}{s}\right)\right] + Z\left[\sum_{k=2}^{n}\left(\frac{1}{s}\right)^k \left(\sum_{k=1}^{k} c_i y_0^{(k-1)}\right)\right] \quad (28)$$

Upon inserting the indicated transforms, and performing the convolutions in equation (28), there results

$$c_1 Z\bar{y} + c_2\left[\frac{T}{2}\,\frac{1+z}{1-z}\,Z\bar{y} - \frac{T}{2}\,\frac{y_0}{(1-z)}\right] + f(k,n,z)\left[Z\bar{y} - \frac{y_0}{2}\right]$$

$$= D(n,z)\left[Z\bar{x} - \frac{x_0}{2}\right] + c_1 y_0\left(\frac{1}{1-z}\right) + g(i,k,n,z) \quad (29)$$

where $D(n,z)$ is defined in equation (13);

$$f(k,n,z) = \sum_{k=2}^{n} c_{k+1} \frac{T^k z}{(k-1)!} \frac{A_k(z)}{(1-z)^k}$$

$$g(i,k,n,z) = \sum_{k=2}^{n} \left[\sum_{i=1}^{k} c_i y_0^{(k-i)} \right] \frac{T^{k-1}}{(k-1)!} \frac{z\, A_k(z)}{(1-z)^k}$$

After multiplying by $(1-z)^n$, and collecting terms such that the left side contains only the terms associated with $Z\bar{y}$, we obtain:

$$\left[A(n,z) + B(n,z(+ C(n,k,z) \right] Z\bar{y} = D(n,z) \left[Z\bar{x} - \frac{x_0}{2} \right] + y_0 \left[F(n,z) + \frac{1}{2} C(n,k,z) \right] + G(k,i,n,z) \qquad (30)$$

where $A(n,z)$, $B(n,z)$, $C(n,k,z)$, and $D(n,z)$ have been defined in equations (10) through (13), and

$$F(n,z) = (c_1 + \frac{c_2 T}{2}) (1-z)^{n-1} \qquad (31)$$

$$G(k,i,n,z) = \sum_{k=2}^{n} \left[\sum_{i=1}^{k} c_i y_0^{(k-i)} \right] \frac{T^{k-1}}{(k-1)!} z(1-z)^{n-k} A_k(z) \qquad (32)$$

The left side of equation (30) is identical to the left side of equation (9) and the same holds true for the forcing function factor. The difference lies in two factors on the right containing the initial conditions.

Collection of z-Coefficients

As was done with equation (9), the coefficients of like powers of z must be collected in equation (30). Since only the right side differs from equation (9), the problem is to

add the coefficients of $F(n,z)$ and $\frac{1}{2}C(n,k,z)$, multiplying them by y_0, and then to add coefficients of $G(k,i,n,z)$. These totaled coefficients are then added to the respective $\frac{T^n}{(n-1)!}w_k$ terms in the recurrence relationship.

It is found that $F(n,z)$, $\frac{1}{2}C(n,k,z)$, and $G(k,i,n,z)$ are all of degree (n-1). Their addition follows the same scheme illustrated in equations (18). Since the result is a polynomial in z, say $R(z)$, of degree (n-1), the final recurrence relation for the n^{th} order differential equation differs from equation (24) only during the first n iterations. To illustrate this, let

$$R(z) = v_0 + v_1 z + v_2 z^2 + \ldots + v_{n-1} z^{n-1}$$

Then, for $k \leq n$

$$y_k = \frac{1}{A_1}\left[\frac{T^n}{(n-1)!}w_k + v_k - A_2 y_{k-1} - A_3 y_{k-2} - \ldots A_{n+1} y_{k-n}\right] \quad (33)$$

For $k > n$, y_k is again given by equation (24).

This concludes the development of the iterated solution for the n^{th} order differential equation, with existing initial conditions for y(t) and its (n-1) derivatives.

The next section will show the development of a FORTRAN program which performs all operations indicated above for any n.

DEVELOPMENT OF THE FORTRAN PROGRAM

This program is designed to solve any ordinary differential equation with constant coefficients of order n such that $2 \leq n \leq 15$.

T. Rado's notation, * for START and ** for STOP, will be used in all flow charts for the sake of notation economy.

Equation (30) is used as the starting point since it is the last step to contain all polynomials separately. These polynomials are a function of n and they must be computed separately before being added to give the final polynomial.

The A(n,z) Polynomial

Equation (10) shows that this merely involves the binomial expansion of $(1-z)^n$. The coefficients of this expansion are determined as shown by the flow chart in Fig. 1, and are denoted by a(k), where (k-1) denotes the power of z with which the a(k) is associated. The constant c_1 is not yet included. Because of memory economy it was found advantageous to expand and store these polynomials as integers; this required delay of multiplication by the floating point number c_1.

The B(n,z) Polynomial

As equation (11) shows, B(n,z) is the expansion of $(1+z)(1-z)^{n-1}$. The coefficients of this expansion are denoted

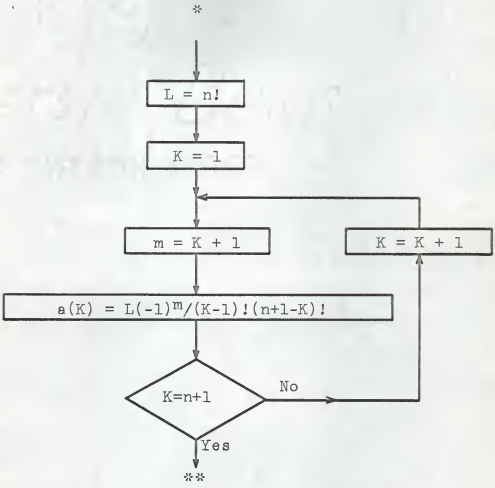

Fig. 1. Expansion of $(1-z)^n$.

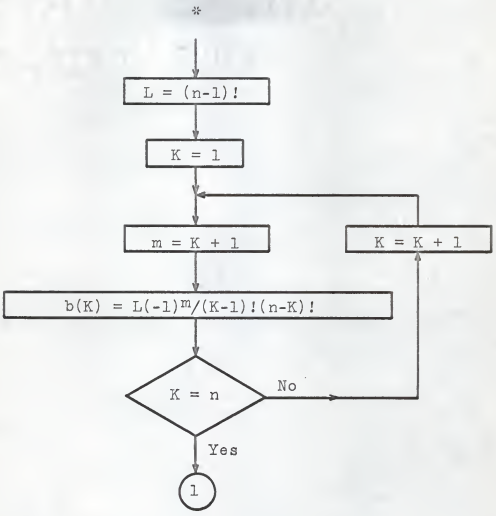

Fig. 2a. Expansion of $b(K) = (1+z)(1-z)^{n-1}$.
Flow chart is continued in Fig. 2b.

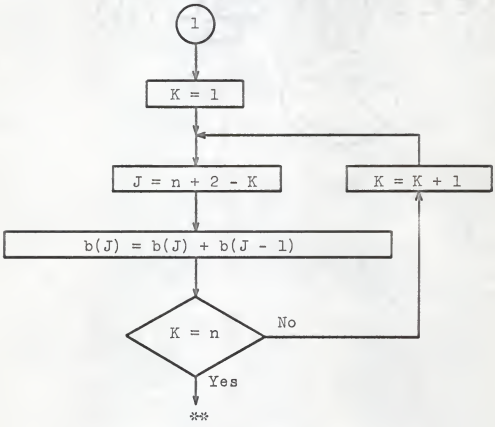

Fig. 2b. Continued expansion of $b(K) = (1+z)(1-z)^{n-1}$.

by b(k), where again, b(k) is the coefficient of z^{k-1}. The flow chart for this expression is shown in Fig. 2a and Fig. 2b.

The C(n,k,z) Polynomials

The next task is expansion of C(n,k,z) polynomials in equation (12). Here again, the integer coefficients of z will be computed and stored. They are denoted by

$c(I,J) = \sum_{k=2}^{n} z\, A_k(z)(1-z)^{n-k}$. The multiplication by

$\sum_{k=2}^{n} \frac{c_{k+1} I^k}{(k-1)!}$ is delayed until the polynomials are summed for reasons of economizing the required memory.

It is noted that these polynomials include the $A_k(z)$ polynomials, derived and computed by Criswell (2). A closed form for these polynomials is available, and a built-in routine for their computation was considered. However, since this program was intended to accept equations of order ≤ 15, it was found that memory space could be saved by inputting and storing coefficients, rather than calculating them in the program.

Initially, the $A_k(z)$ coefficients were read-in and stored in a square array. As the program progressed, however, memory capacity was exceeded on several occasions. One of the modifications made to conserve memory space was to store the $A_k(z)$ polynomials in a vector, in which each polynomial occupies a minimum number of positions. It will be noted that the $A_k(z)$ array (2) has zeros in its upper triangle. These positions

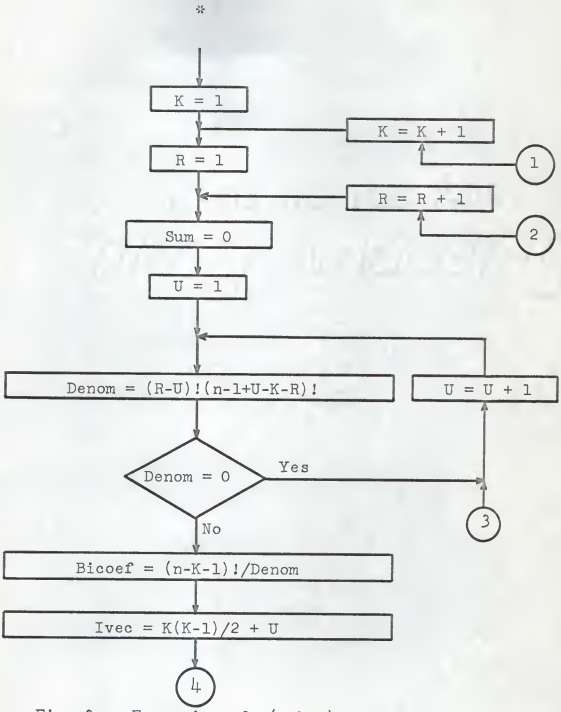

Fig. 3a. Expansion of $c(n,k,z) = \sum_{k=2}^{n} z\, A_k(z)(1-z)^{n-k}$.

Continued in Figs. 3b and 3c.

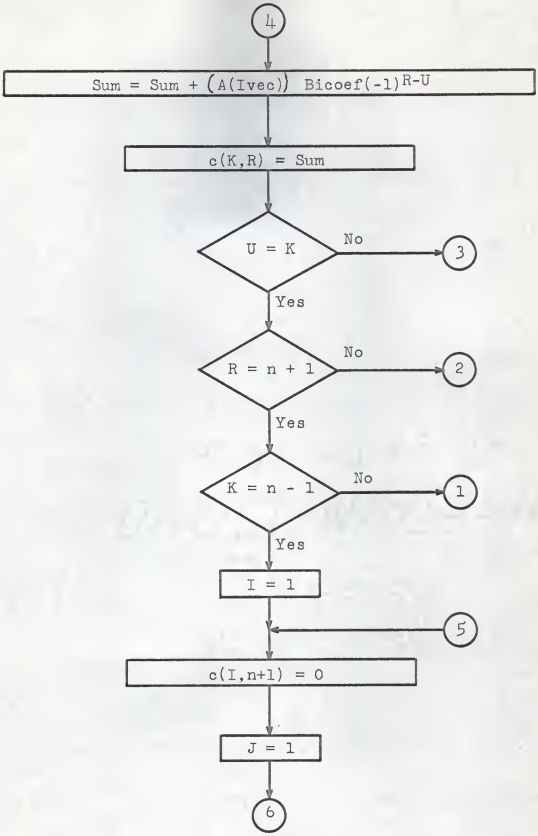

Fig. 3b. Expansion of $c(n,k,z) = \sum_{k=2}^{n} z\, A_k(z)(1-z)^{n-k}$.

Continued in Fig. 3c.

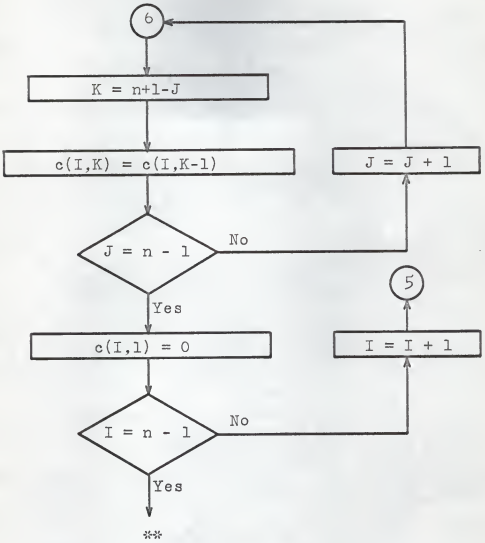

Fig. 3c. Expansion of $c(n,k,z) = \sum_{k=2}^{n} z\, A_k(z)(1-z)^{n-k}$.

were eliminated by the vector form. The following transformation was used to achieve this. Let A(i,j) be a lower triangular matrix, and let it be stored by rows in a vector B(k). The A(i,j) element can be stored in the B $\left(\left[i(i-1)/2 \right] + j \right)$ element.[1] The B address is denoted by IVEC in the program.

Summing A(n,z), B(n,z), and C(n,k,z) Polynomials

At this point the above polynomials have been expanded and the summation indicated by equations (18) is to be performed. The a(n,z), b(n,z), and c(n,k,z) are converted to floating point numbers, multiplied by their respective coefficients, and summed into the single polynomial A(z) of equation (19). In the program this polynomial is denoted as A12T(K).

The D(n,z) Polynomial

Expansion of the D(n,z) polynomial involves recall of the proper $A_k(z)$ polynomial stored in vector form along with all other $A_k(z)$ polynomials. After recall, the coefficient is multiplied by z, which causes a shift of "one" in its position subscript. The resulting polynomial d(k) is of degree n, but its z^0 term and its z^n term have zero coefficients. Multiplication by the constant $T^n/(n-1)!$ is again delayed until the coefficients are used in the W(K) subroutine. The flow chart is shown in Fig. 5.

---

[1] Mr. David G. Dutra can be credited for this innovation.

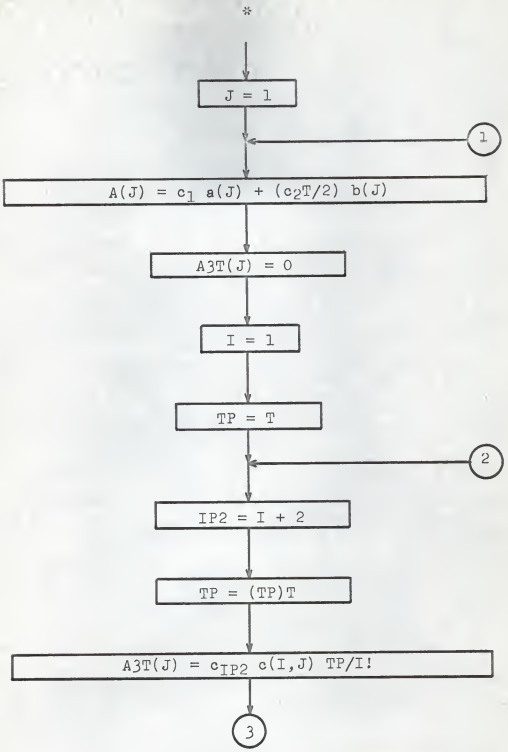

Fig. 4a. Summation of the polynomials A(n,z), B(n,z), and C(n,k,z). Continued in Fig. 4b.

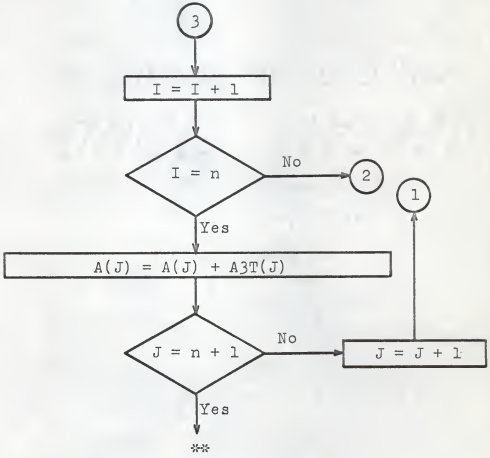

Fig. 4b. Summation of A(n,z), B(n,z), and C(n,k,z).

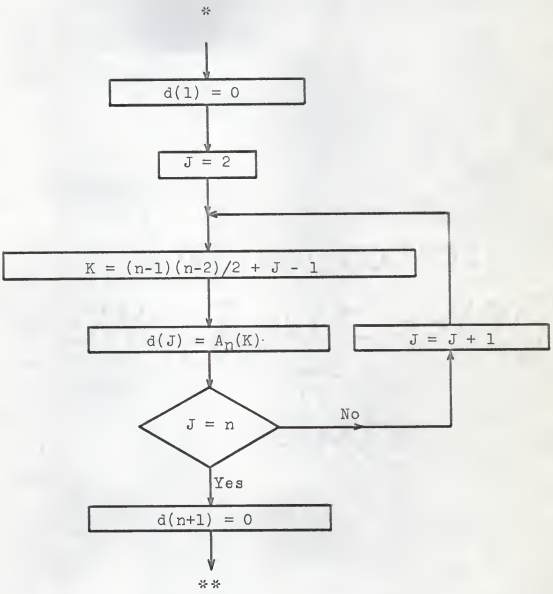

Fig. 5. Expansion of $d(n,z) = z\, A_n(z)$.

Expanding the F(n,z) Polynomial

Equation (31) shows that F(n,z) requires the binomial expansion of $(1-z)^{n-1}$; these binomial coefficients are denoted by f(k) in Fig. 6. Multiplication by the constant $c_1 + (c_2T/2)$ is performed simultaneously with the addition $F(n,z) + (1/2)C(n,k,z)$.

Addition of F(n,z) and (1/2)C(n,k,z)

Coefficients of C(n,k,z) are already available and are denoted by A3T(k) in Fig. 4a. To perform the addition, recall A3T(k), divide by 2, and add this to $c_1 + (c_2T/2)$ times the f(k) coefficients. This operation is illustrated in Fig. 7. Coefficients of the resulting polynomial are denoted by FC(k).

Expansion of the G(k,i,n,z) Polynomial

The G(k,i,n,z) polynomial is defined in equation (32). Its expansion is performed in two distinct steps. The factor contained in the brackets of equation (32), $\sum_{i=1}^{k} c_i y_0^{(k-i)}$, is a constant which depends on coefficients of derivative terms in equation (2) and the initial values of y(t), and its (n-1) derivatives.

Summation of this factor is performed in a subroutine which is denoted as the CO(K) subroutine. The argument K, of

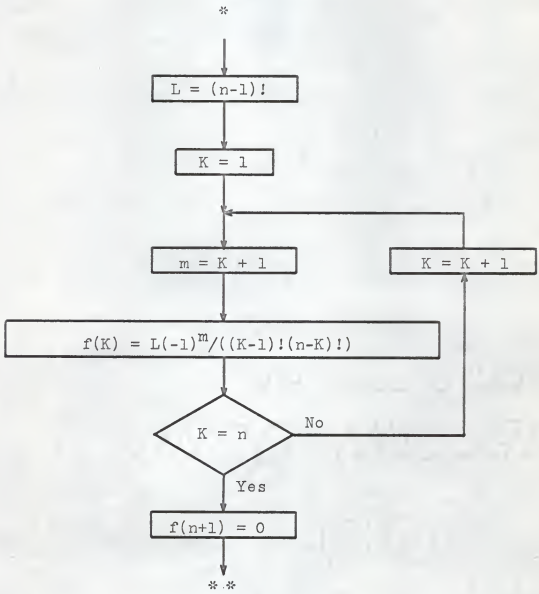

Fig. 6. Expansion of $(1-z)^{n-1}$.

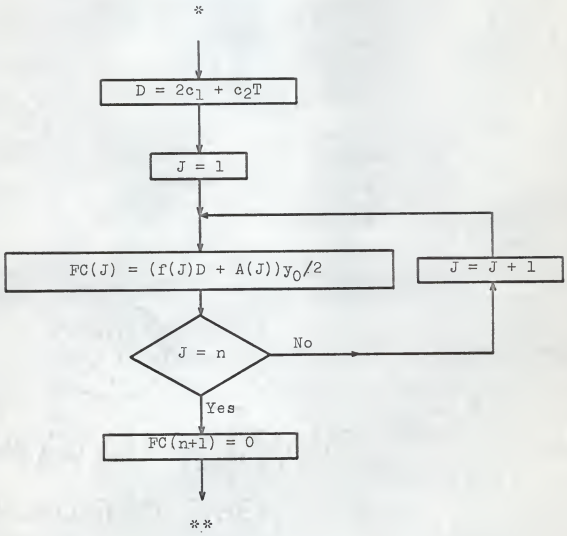

Fig. 7. The summation F(n,z) + (1/2)C(n,k,z).

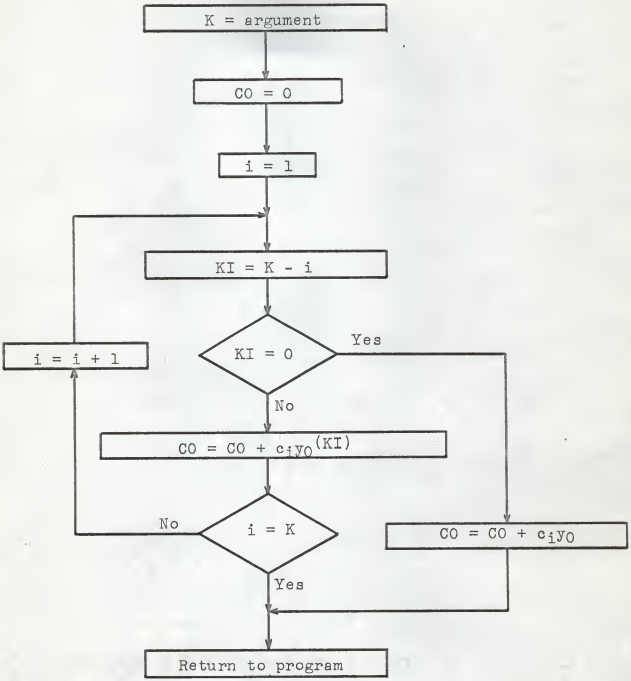

Fig. 8. Calculation of $CO(K) = \sum_{i=1}^{k} c_i y_0^{(k-i)}$.

the subroutine, is synonomous to the upper limit on the summation symbol in equation (32). Thus for every k in equation (32) the subroutine is called and it computes the value of $\sum_{i=1}^{k} c_i y_0^{(k-i)}$. The flow chart for the CO(K) subroutine is shown in Fig. 8.

Once the bracketed term in equation (32) have been determined, the remaining expansion of the G(k,i,n,z) polynomial follows the expansion for C(n,k,z), which has previously been discussed. The flow chart for G(k,i,n,z) may be found in Fig. 9a and Fig. 9b.

The last step in obtaining all polynomials is summation of the F(n,z); 1/2 C(n,k,z) and G(k,i,n,z) polynomials. The resultant polynomial contains all initial conditions, and is of degree n. Its coefficients are added to W_k during the first n iterations. The flow chart for this polynomial is shown in Fig. 10.

Determining the W(K) Coefficients

The W(K) coefficients are entries in the sequence W_k. They are defined in equation (21), and must be determined for every k, where $1 \leq k \leq \left[t_s/T\right]$, t_s is a prespecified solution time, and T is the sampling interval. Equation (33) shows how the W(K) become part of the solution recurrence relation.

The W(K) are generated in a subroutine whose argument is K. This subroutine determines the value of W(K) for the particular

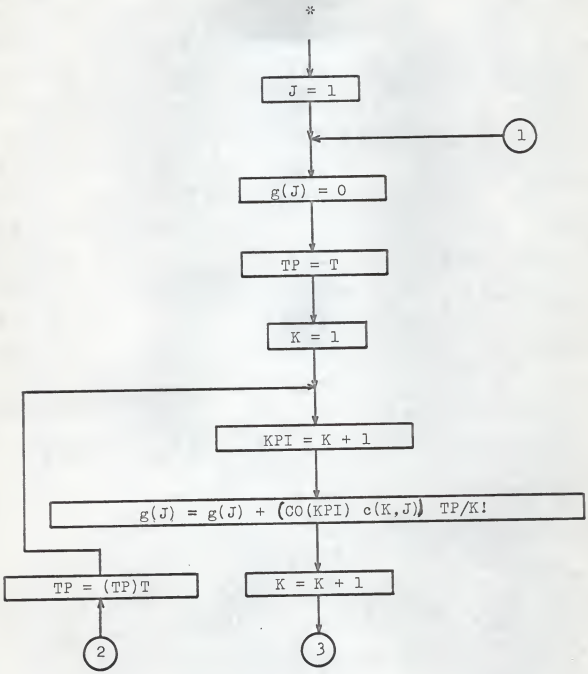

Fig. 9a. Expansion of G(k,i,n,z).
Continued in Fig. 9b.

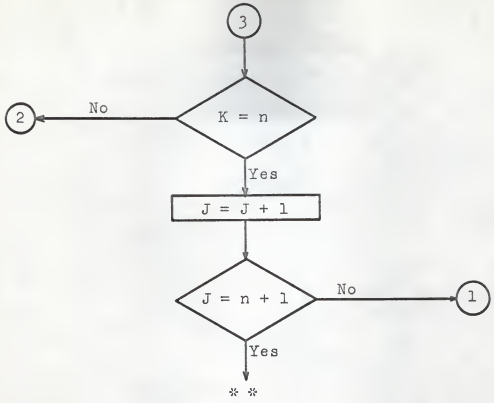

Fig. 9b. Expansion of G(k,i,n,z).

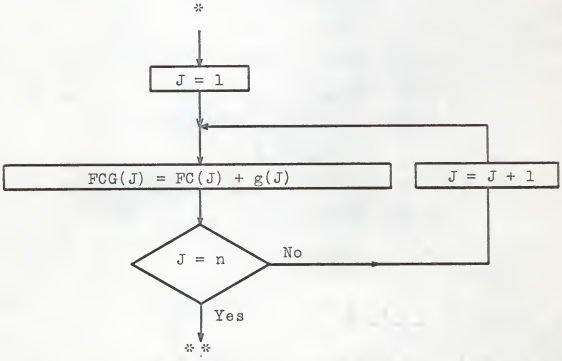

Fig. 10. The summation F(n,z) + (1/2)C(n,k,z) + G(i,k,n,z).

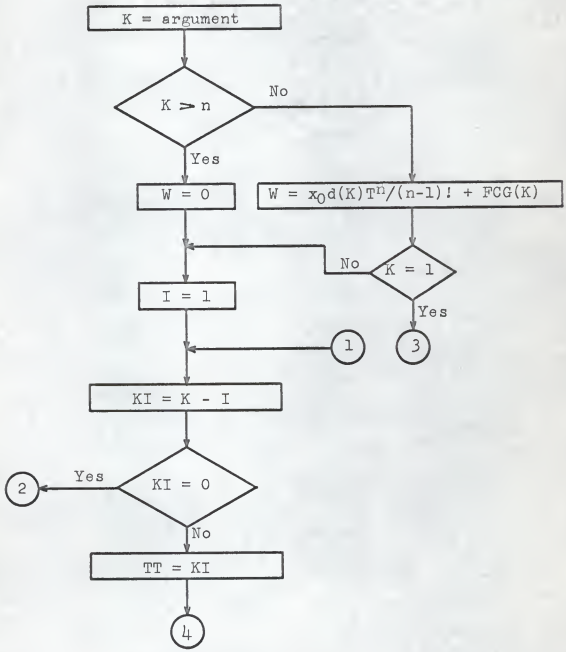

Fig. 11a. The W(K) subroutine.
Continued in Fig. 11b.

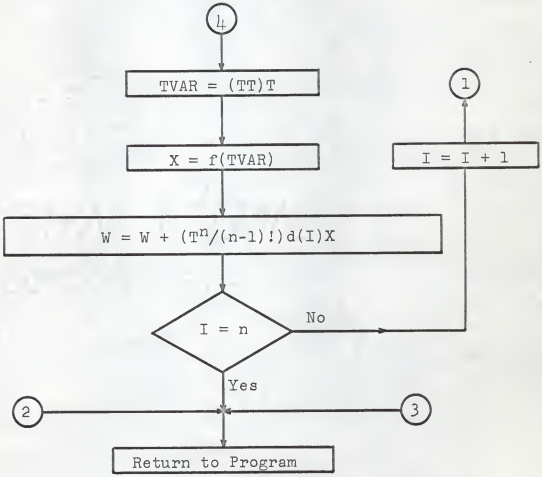

Fig. 11b. The W(K) subroutine.

K, and returns W(K) to the main program. The flow chart of the W(K) subroutine is shown in Fig. 11a and Fig. 11b.

The Iterated Solution

As equation (33) points out, the present value to be calculated is always dependent on n previously calculated values. Therefore the n previous y_k's must be available in storage. Since k may become a very large number, it is impossible to store all y_k's. Thus the program was designed to calculate, store, and print n of the y_k's. After they are printed out, they are replaced with new y_k's, until all n of them are replaced and at this time they are printed again. By using this method, the core storage required for the output was reduced to n positions, which is minimal. The flow chart for the iteration is shown in Fig. 12a, Fig. 12b, and Fig. 12c.

Other Program Features

It was pointed out earlier that the A(1) coefficient could attain a value of zero. A check feature to detect a zero is included. The program remedies this defect by choosing one-half the requested sampling time.

Since it is very common to solve differential equations whose initial conditions are zero, a feature has been included which scans the initial conditions. If it finds all of them to be zero, the routines for the $\left[F(n,z) + (1/2)C(n,k,z)\right]$ and the

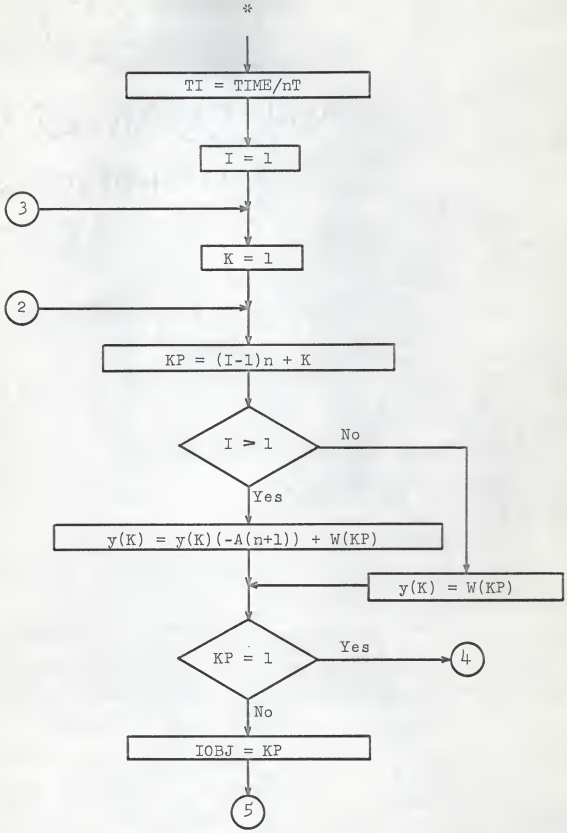

Fig. 12a. Iterated solution. Continued in Fig. 12b and Fig. 12c.

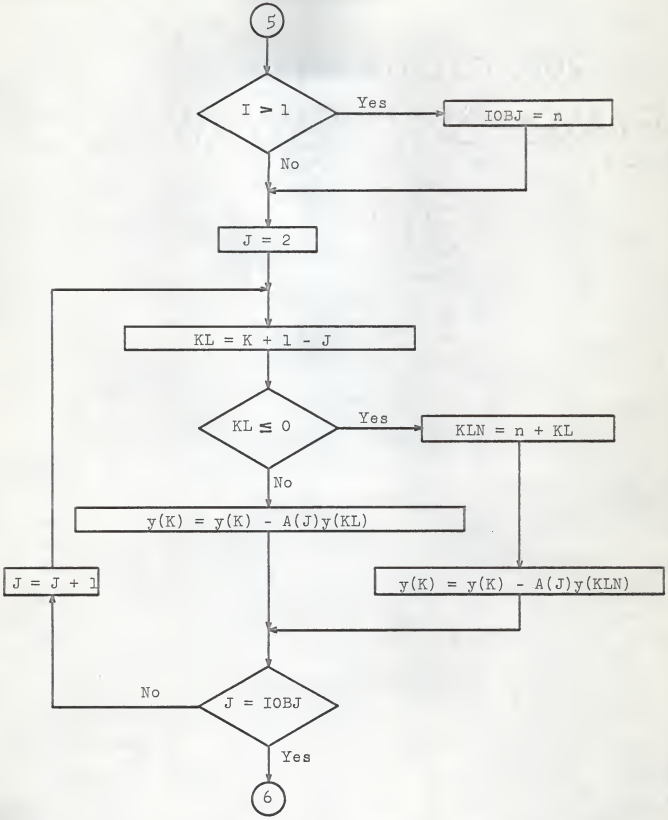

Fig. 12b. Iterated solution. Continued in Fig. 12c.

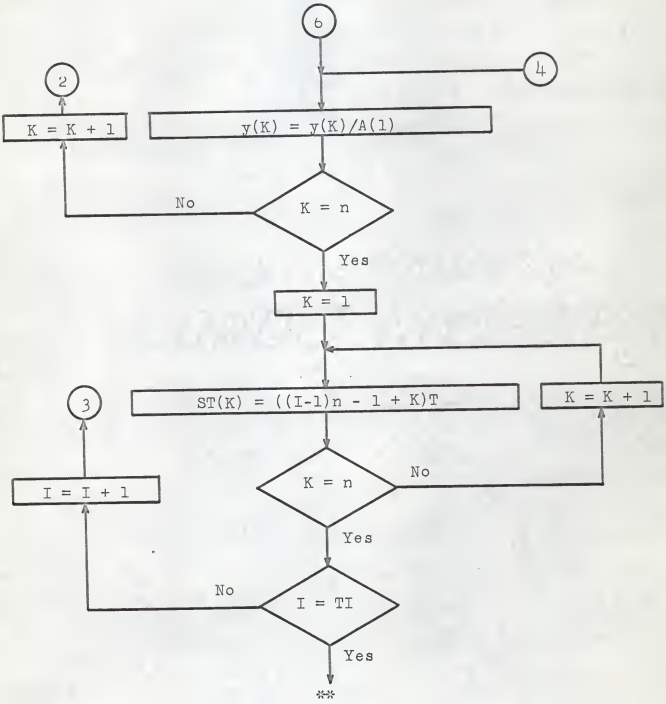

Fig. 12c. Iterated solution.

G(k,i,n,z) coefficients are by-passed to save computation time.

Factorials have been described in regular mathematical notation in the flow charts. However, the program calculates them in one of two incorporated subroutines: one subroutine for floating point numbers, and the other subroutine for fixed point numbers.

The output format statements are written for a 132-character line printer, and adapt to any order of the input equation. Discrete time for each iteration is listed to the left of the computed output.

This concludes the outline of the FORTRAN program. A listing of the program is found in the appendix. It should be noted that notations used in the flow charts and the program do not always agree, for the sake of simplicity, in the flow charts.

The program in its present form is geared to the IBM 1410-PR-155 computer. It is written in FORTRAN IV language, and requires approximately 38,000 digits of core storage. Work tapes and CHAIN features are not used and they were purposely avoided for the sake of increased computation speed. Computer time required is a function of the solution time requested, and the order of the equation. Fifteen minutes of computation time (not including compilation) was rarely exceeded. In order to reduce compilation time it is recommended that an object deck be punched for the entire program except for the W(K) subroutine. This would require that the source deck for the subroutine be placed ahead of the object deck, and that an EXEQ FORTRAN card precede it. This technique reduces compilation tome to a

fraction of a minute.

The program operates with 18 significant figures for both floating and fixed point numbers; this was the maximum available in the available digital computer.

All subroutines are FUNCTION subroutines, which share a number of variables with the main program in a COMMON memory area. It should be mentioned that this COMMON area does not appear on the memory map, and therefore must be added to the last point of entry, plus the size of the last routine loaded, in order to arrive at the total size of the program. However, if a total capacity of 40,000 digits of core storage is available, it is doubtful that this capacity would ever be exceeded, since this is possible only by a forcing function which requires more than 2,000 digits. Should this case arise, it is recommended that the fixed point numbers be reduced to 12 significant figures or less to make up the deficit. This will result in little or no loss of accuracy in the output.

USING THE FORTRAN PROGRAM

The FORTRAN program is an auto-program in the sense that no additional programming is required in order to obtain a solution for an ordinary differential equation with constant coefficients, provided its order n is $2 \leq n \leq 13$. Therefore the only input requirements are a statement of the problem to be solved, the desired sampling time, and solution time.

Entering the Problem into the Program

To aid in identifying the various quantities, equation (1) is repeated for convenience:

$$c_1 \frac{d^{(n)}y(t)}{dt^n} + c_2 \frac{d^{(n-1)}y(t)}{dt^{n-1}} + \ldots + c_{n+1}y(t) = x(t) \quad (1)$$

Input Data Cards. The first input data card to be placed on the back of the deck is a card containing n, X_{not}, Time, and T, and in that order from left to right. Here

- n denotes the highest order of derivative contained in the equation and is an integer,
- X_{not} denotes the initial value of the forcing function,
- Time denotes the desired solution time,
- T denotes the sampling interval.

The FORTRAN format for this card is (I3, E24.18, 2F10.5).

Example. Let n = 4, X_{not} = 1.0, Time = 5.0, T = .02. The card appears as follows:

4	.1 E 1	5.0	.02
↑	↑	↑	↑
Column 3	Column 27	Column 37	Column 47

The next input card is the card containing c_1, followed by cards bearing the remaining coefficients of the derivative, one coefficient per card. The FORTRAN format is (E24.18). It is important that there are exactly (n+1) cards to be read by the program for coefficients of the derivatives. If any term on the left side of equation (1) is absent, the coefficient for that term must be read-in as zero, i.e., a blank card is placed

in the proper position of the sequence of the c_i input cards.

The last set of input cards to be placed on the back of the deck is the set of cards containing the initial value of y(t) and its (n-1) derivatives. The FORTRAN format is again (E24.18), with one number per card. The first card must contain y_0 and it is placed immediately following c_{n+1}. The second card must contain $y_0^{(1)}$, and so on in ascending order of derivatives, up to $y_0^{(n-1)}$. Thus $y_0^{(n-1)}$ is the last card of the deck. If all initial conditions are zero, it is required that n blank cards be read-in.

The forcing function input card cannot be read-in by the program; hence it must be prepared in proper FORTRAN language and placed in the proper position of the deck. The position is in the W(K) subroutine, and is denoted by a comment card. The discrete time is available in the W(K) subroutine, and is denoted by TVAR. Thus if the forcing function is a time-varying quantity, then the time variable must be TVAR.

Example. Let the forcing function be:

$$x = 3t^2 + 5 \sin 6t$$

The card containing this forcing function appears as follows:

```
      X = 3.*TVAR**2 + 5.*SIN(6.*TVAR)
      ↑
Column 7
```

This concludes the discussion of all input data required. As a check-feature, it is noted that for an n^{th} order differential equation, there are always 2n+3 input cards required; this assumes that only one card is required for the forcing function.

Choice of Sampling Time

Choice of a suitable sampling time is very important, and yet no universal and satisfactory method for determining such a sampling time has been found. It is, of course, dependent on the smallest time constant of the equation, as well as on the period of the forcing function. The sampling time must be a fraction of the smallest of the two mentioned above. Since the forcing function period is oftentimes least important, the quantity to be considered is the smallest time constant of the equation. This is determined by the smallest root of the polynomial in s associated with the left side of equation (1).

Consider the special case where all derivatives are present, and the s polynomial on the left side of equation (3) is normalized such that $c_{n+1} = 1$. Then it can be shown that c_n is the sum of all time constants in the polynomial factors. Knowing that there are n roots, a division by n results in an average time constant. This average value may then be decreased by a factor of 10 or 20 in hopes of approaching the smallest time constant. This results in $T = c_n/10(n)c_{n+1}$. Experience has shown that this sampling interval gives fair results.

A word of warning against choosing an arbitrarily small sampling time is appropriate at this time. The mathematical development contains the coefficient $T^n/(n-1)!$ in several places. Thus when choosing T very small, this coefficient becomes extremely small and any time that quantities associated with this coefficient become added or subtracted, these

quantities may be lost due to limited register sizes in the digital computer.

Unit Impulse Forcing Function

It is frequently the case that the unit impulse function, $\delta(t)$, is used as a forcing function on differential equations with constant coefficients. In order to solve such a problem with the FORTRAN program developed in this paper, it is merely necessary to apply a forcing function of zero, and increase the value of $y_0^{(n-1)}$ by "one".

Example. Suppose that the response of an equation to a unit impulse function is to be found and that all initial conditions are zero. The n, X_{not}, Time, T card, and the c_i cards are prepared in the usual manner. The $y_0^{(i)}$ cards are added as blank cards, except the card for $y_0^{(n-1)}$; it must bear the value "one". The forcing function X, in the W(K) subroutine, is given the value "zero".

ERROR ANALYSIS

Throughout development of the FORTRAN program, a family of differential equations was used for testing purposes; it is listed in equation (34).

$$\prod_{k=1}^{n} (D + k)y(t) = n! \tag{34}$$

Here $D = \frac{d}{dt}$ and all initial conditions are zero. The exact solution to this equation is:

$$y(t) = (1 - e^{-t})^n \tag{35}$$

Error analyses using the above equations, as well as others, have been performed and a few typical results are listed. In all cases, the error column shows the quantity: program output minus exact solution.

A 6^{th}-order Differential Equation

A variation of equation (34), which uses initial conditions, is as follows:

$$\prod_{k=1}^{n} (D + k)y(t) = 0 \tag{36}$$

where $y_0^{(k)} = 0, \quad k = 0, 1, 2, \ldots, (n-2)$

$y_0^{(n-1)} = n!$

The exact solution to these equations is:

$$y(t) = n(1 - e^{-t})^{n-1}(e^{-t}) \tag{37}$$

An error analysis, using equations (36) and (37), for the case n = 6, is listed in Table 1. The sampling interval was

TABLE 1. ERROR ANALYSIS OF EQ.(36), T=.02 SEC.

TIME	PROGRAM OUTPUT	EXACT SOLUTION	ERROR
.000	.0000000000	.0000000000	.0000000000
.100	.0000416663	.0000423694	-.0000007030
.200	.0009586242	.0009614196	-.0000027954
.300	.0051985853	.0051985860	-.0000000007
.400	.0156733710	.0156637651	.0000096059
.500	.0343406003	.0343208863	.0000197139
.600	.0615919820	.0615692714	.0000227105
.700	.0963483956	.0963328839	.0000155117
.800	.1365143404	.1365138878	.0000004525
.900	.1795120918	.1795291398	-.0000170479
1.000	.2227383861	.2227698155	-.0000314294
1.100	.2638833526	.2639223361	-.0000389834
1.200	.3011097240	.3011482471	-.0000385230
1.300	.3331179557	.3331489991	-.0000310434
1.400	.3591310605	.3591499461	-.0000188856
1.500	.3788308058	.3788356438	-.0000048379
1.600	.3922704486	.3922618815	.0000085670
1.700	.3997818270	.3997623597	.0000194673
1.800	.4018880759	.4018612610	.0000268149
1.900	.3992281045	.3991977830	.0000303214
2.000	.3924953274	.3924650383	.0000302891
2.100	.3823907736	.3823633734	.0000274002
2.200	.3695893150	.3695667982	.0000225167
2.300	.3547170790	.3547005606	.0000165183
2.400	.3383379045	.3383277146	.0000101899
2.500	.3209467697	.3209426123	.0000041574
2.600	.3029683529	.3029694890	-.0000011360
2.700	.2847591769	.2847646021	-.0000054252
2.800	.2666120955	.2666206944	-.0000085988
2.900	.2487621583	.2487728252	-.0000106669
3.000	.2313931400	.2314048661	-.0000117260
3.100	.2146442239	.2146561498	-.0000119259
3.200	.1986164895	.1986279320	-.0000114424
3.300	.1833789847	.1833894407	-.0000104559
3.400	.1689742527	.1689833884	-.0000091357
3.500	.1554232548	.1554308855	-.0000076306
3.600	.1427296775	.1427357416	-.0000060640
3.700	.1308836413	.1308881733	-.0000045319
3.800	.1198648528	.1198679574	-.0000031046
3.900	.1096452484	.1096470770	-.0000018286
4.000	.1001911861	.1001919167	-.0000007305
4.100	.0914652402	.0914650612	.0000001789
4.200	.0834276525	.0834267508	.0000009016
4.300	.0760374916	.0760360433	.0000014482
4.400	.0692535636	.0692517283	.0000018352
4.500	.0630351166	.0630330339	.0000020826
4.600	.0573423727	.0573401608	.0000022118

.02 second. The program output is seen to maintain 4-figure accuracy throughout most of the solution.

A 13^{th}-order Differential Equation

To demonstrate the ability of the program to solve high order equations, an error analysis of equation (34), for n = 13, is shown in Table 2. The sampling time used in the solution was 0.02 second, and only sample outputs are listed in the table.

Initially, while the output values are very small, the error is approximately four per cent; this is caused by limited register sizes. At Time = 3.0 seconds, the error is less than 0.02 per cent; however, as time becomes larger, the error increases to about 0.3 per cent due to recursively used approximate output values.

A 3^{rd}-order Differential Equation with Unbounded Solution

To determine the behavior of the program in cases of unbounded solutions, an error analysis of the following equation was performed.

$$\dddot{y} - 3\ddot{y} - 4\dot{y} + 12y = 12\, e^{-t} \tag{38}$$

$$y(0) = 4; \quad \dot{y}(0) = 2; \quad \ddot{y}(0) = 18$$

The exact solution is:

$$y(t) = e^{-2t} + e^{2t} + e^{3t} + e^{-t} \tag{39}$$

The error analysis is shown in Table 3. In this problem, the initial value of the response is large enough such that no

TABLE 2. ERROR ANALYSIS OF EQ.(34), N=13, T=.02 SEC.

TIME	PROGRAM OUTPUT	EXACT SOLUTION	ERROR
.000	.0000000000	.0000000000	.0000000000
.100	.0000000000	.0000000000	.0000000000
.200	.0000000002	.0000000002	.0000000000
.400	.0000005564	.0000005435	.0000000129
.500	.0000054895	.0000054180	.0000000714
.600	.0000323561	.0000321109	.0000002452
.700	.0001339868	.0001333633	.0000006234
.800	.0004294542	.0004281530	.0000013011
.900	.0011342541	.0011318964	.0000023576
1.000	.0025765706	.0025727577	.0000038128
1.100	.0051903205	.0051847395	.0000055809
1.200	.0094836707	.0094762043	.0000074663
1.300	.0159894190	.0159802007	.0000092182
1.400	.0252080129	.0251974015	.0000106113
1.500	.0375543704	.0375428637	.0000115067
1.600	.0533170796	.0533052116	.0000118679
1.700	.0726345915	.0726228517	.0000117398
1.800	.0954891232	.0954779081	.0000112151
1.900	.1217160228	.1217056117	.0000104110
2.000	.1510246406	.1510151747	.0000094659
2.100	.1830261981	.1830176439	.0000085542
2.200	.2172644486	.2172565350	.0000079135
2.300	.2532457268	.2532378546	.0000078722
2.400	.2904659797	.2904571096	.0000088700
2.500	.3284333388	.3284218763	.0000114625
2.600	.3666856216	.3666693192	.0000163024
2.700	.4048027829	.4047786709	.0000241120
2.800	.4424147690	.4423791000	.0000356690
2.900	.4792054630	.4791536298	.0000518332
3.000	.5149134904	.5148398733	.0000736171
3.100	.5493306320	.5492283485	.0001022835
3.200	.5822985126	.5821590769	.0001394357
3.300	.6137041328	.6135170709	.0001870618
3.400	.6434747069	.6432272075	.0002474993
3.500	.6715721888	.6712488735	.0003233153
3.600	.6979877934	.6975706713	.0004171220
3.700	.7227367580	.7222053858	.0005313722
3.800	.7458535142	.7451853414	.0006681728
3.900	.7673873665	.7665582255	.0008291410
4.000	.7873987180	.7863834082	.0010153098
4.100	.8059558478	.8047287610	.0012270867
4.200	.8231322245	.8216679538	.0014642707
4.300	.8390043242	.8372781959	.0017261283
4.400	.8536498954	.8516383811	.0020115143
4.500	.8671466011	.8648275903	.0023190108
4.600	.8795709571	.8769239071	.0026470499
4.700	.8909975011	.8880035026	.0029939984

TABLE 3. ERROR ANALYSIS OF EQ.(38), T=.005 SEC.

TIME	PROGRAM OUTPUT	EXACT SOLUTION	ERROR
.000	4.0000000000	4.0000000000	.0000000000
.050	4.1230723546	4.1230720033	.0000003512
.100	4.2948308044	4.2948297368	.0000010676
.150	4.5196994790	4.5196971901	.0000022888
.200	4.8029984846	4.8029942971	.0000041874
.250	5.1510597066	5.1510527300	.0000069765
.300	5.5713626873	5.5713517683	.0000109190
.350	6.0726935577	6.0726772190	.0000163386
.400	6.6653304951	6.6653068613	.0000236337
.450	7.3612597460	7.3612264532	.0000332928
.500	8.1744269140	8.1743809996	.0000459143
.550	9.1210289748	9.1209667452	.0000622296
.600	10.2198533661	10.2197702351	.0000831310
.650	11.4926715227	11.4925618180	.0001097047
.700	12.9646954182	12.9645521471	.0001432710
.750	14.6651070516	14.6649216195	.0001854320
.800	16.6276724156	16.6274342871	.0002381284
.850	18.8914533382	18.8911496303	.0003037078
.900	21.5016327431	21.5012477372	.0003850058
.950	24.5104713688	24.5099859255	.0004854433
1.000	27.9784168882	27.9778077465	.0006091416
1.050	31.9753897317	31.9746286708	.0007610608
1.100	36.5822738251	36.5813266621	.0009471629
1.150	41.8926449844	41.8914703766	.0011746078
1.200	48.0147749740	48.0133229895	.0014519845
1.250	55.0739553438	55.0721657562	.0017895876
1.300	63.2151922564	63.2129925117	.0021997446
1.350	72.6063317491	72.6036345437	.0026972053
1.400	83.4416844425	83.4383848385	.0032996039
1.455	97.2981833086	97.2940746025	.0041087060
1.500	110.3804930143	110.3755854522	.0049075621
1.550	127.0462023071	127.0402340347	.0059682723
1.600	146.2928523430	146.2856064378	.0072459052
1.650	168.5253189952	168.5165359181	.0087830770
1.700	194.2126946745	194.2020641413	.0106305332
1.750	223.8985404090	223.8856917441	.0128486648
1.800	258.2127825616	258.1972732585	.0155093031
1.850	297.8855188009	297.8668209586	.0186978423
1.900	343.7630406016	343.7405248514	.0225157502
1.950	396.8264291030	396.7993455672	.0270835357
2.000	458.2131387052	458.1805944480	.0325442572

accuracy is lost due to register sizes, and the percentage error is extremely small for the first few iterations. Nonetheless, the error increases with solution time because of recursively used approximate output values. It is noted, however, that at Time = 2.0 seconds, the error is still only about 0.007 per cent.

The sampling time used for the output in Table 3 is 0.005 second; the error for a sampling time of 0.01 second was slightly larger.

A 2nd-order Differential Equation with Oscillatory Solution

Various monotonic responses have been investigated, and this problem demonstrates the ability of the program to determine oscillatory responses. The equation solved for this purpose is

$$\ddot{y} + 2\dot{y} + 2y = -2 \cos 2x - 4 \sin 2x \qquad (40)$$

$$y(0) = 1; \quad \dot{y}(0) = 1$$

The exact solution is

$$y(t) = e^{-x} \sin x + \cos 2x \qquad (41)$$

The sampling time used in this error analysis is 0.02 second, and the result is listed in Table 4. It is noted that the error remains extremely small for the entire solution time. The maximum error is approximately 0.04 per cent, and it remains below 0.03 per cent for iterations beyond Time = 1.0 second.

TABLE 4. ERROR ANALYSIS OF EQ.(40), T=.02 SEC.

TIME	PROGRAM OUTPUT	EXACT SOLUTION	ERROR
.000	1.0000000000	1.0000000000	.0000000000
.120	1.0776206343	1.0775131780	.0001074563
.240	1.0741546312	1.0739784313	.0001761999
.360	.9977896616	.9975791219	.0002105397
.480	.8594767402	.8592612705	.0002154696
.600	.6724365978	.6722401141	.0001964836
.720	.4515400592	.4513806853	.0001593738
.840	.2125935350	.2124835188	.0001100162
.960	-.0284328750	-.0284870273	.0000541523
1.060	-.2197690140	-.2197755496	.0000065355
1.200	-.4567250049	-.4566689375	-.0000560673
1.320	-.6181413456	-.6180398896	-.0001014560
1.440	-.7312110442	-.7310753081	-.0001357360
1.560	-.7897996881	-.7896430631	-.0001566250
1.680	-.7911428610	-.7909799917	-.0001628693
1.800	-.7359367386	-.7357824856	-.0001542530
1.920	-.6282402201	-.6281086640	-.0001315560
2.040	-.4751935684	-.4750971041	-.0000964643
2.160	-.2865689552	-.2865175194	-.0000514357
2.280	-.0741778762	-.0741783472	.0000004709
2.400	.1488314154	.1487756207	.0000557947
2.520	.3687470130	.3686361243	.0001108887
2.640	.5720934809	.5719313380	.0001621429
2.760	.7464139042	.7462077046	.0002061996
2.880	.8809897522	.8807495998	.0002401523
3.000	.9674579532	.9671962381	.0002617151
3.120	1.0002904104	1.0000210558	.0002693545
3.240	.9771090071	.9768466300	.0002623771
3.360	.8988185209	.8985775540	.0002409669
3.480	.7695502378	.7693440654	.0002061723
3.600	.5964198515	.5962600087	.0001598428
3.720	.3891138236	.3890093048	.0001045187
3.820	.1986452790	.1985915229	.0000537560
3.960	-.0799080286	-.0798876064	-.0000204221
4.080	-.3149852375	-.3149022465	-.0000829910
4.200	-.5324993048	-.5323584355	-.0001408692
4.300	-.6913342249	-.6911510988	-.0001831260
4.440	-.8668332198	-.8666034120	-.0002298077
4.560	-.9645102035	-.9642544226	-.0002557808
4.680	-1.0074439736	-1.0071767887	-.0002671849
4.800	-.9931493972	-.9928860385	-.0002633586
4.920	-.9224136756	-.9221691642	-.0002445114
5.040	-.7992537976	-.7990420853	-.0002117122
5.160	-.6306884420	-.6305216123	-.0001668296
5.280	-.4263374362	-.4262250112	-.0001124249
5.400	-.1978717660	-.1978201598	-.0000516062
5.520	.0416543010	.0416421511	.0000121499
5.640	.2785506569	.2784754598	.0000751970

A 3rd-order Differential Equation

The error analysis of this particular problem was prompted by remarks in reference (6). The error analysis is performed on the following equation.

$$\dddot{y} + 6\ddot{y} + 11\dot{y} + 6y = 6 \qquad (42)$$

$$y(0) = \dot{y}(0) = \ddot{y}(0) = 0$$

The exact solution is

$$y(t) = 1 - 3e^{-t} + 3e^{-2t} - e^{-3t} \qquad (43)$$

Table 5 shows results of this error analysis. A sampling time of 0.04 second was used for the solution. Again, the error is larger for initial iterations because of limited register sizes, but at Time = 1.0 second the error is down to 0.036 per cent, and remains below 0.01 per cent for most of the solution.

This error analysis compares favorably with that listed in reference (6), Table I, which was obtained using Wasow's method. It should be pointed out that the sampling time used in obtaining Table 5, was determined by the algorithm mentioned under "Choice of Sampling Time" in this thesis. This algorithm is an attempt at paying heed to the smallest time constant of the equation, rather than to choose the sampling interval arbitrarily, which is a much too common practice. In this particular example the algorithm insured that there were at least eight samplings during the smallest time constant interval of one-third, as compared to three samplings used in reference (6). If a continuous function is to be approximated by discrete samplings, it is important that samplings are frequent enough

TABLE 5. ERROR ANALYSIS OF EQ.(42), T=.04 SEC.

TIME	PROGRAM OUTPUT	EXACT SOLUTION	ERROR
.000	.0000000000	.0000000000	.0000000000
.240	.0097657726	.0097143362	.0000514363
.480	.0554593666	.0554007238	.0000586428
.720	.1352839404	.1352013871	.0000825533
1.000	.2526723375	.2525804578	.0000918797
1.240	.3589154098	.3588430551	.0000723547
1.480	.4607842444	.4607477478	.0000364965
1.720	.5532511619	.5532539125	-.0000027505
2.000	.6464219161	.6464623147	-.0000403986
2.240	.7133572346	.7134181881	-.0000609535
2.480	.7691513078	.7692218215	-.0000705136
2.720	.8150372311	.8151083241	-.0000710929
3.000	.8578875889	.8579516416	-.0000640526
3.240	.8869953242	.8870496767	-.0000543524
3.480	.9103523973	.9103958174	-.0000434200
3.720	.9290130601	.9290457198	-.0000326596
4.000	.9460318826	.9460533270	-.0000214443
4.240	.9573835281	.9573969698	-.0000134417
4.480	.9663766081	.9663836435	-.0000070354
4.720	.9734900539	.9734921969	-.0000021430
5.000	.9799239444	.9799220528	.0000018915
5.240	.9841875248	.9841833587	.0000041660
5.480	.9875496691	.9875440883	.0000055807
5.720	.9901994385	.9901931015	.0000063370
6.000	.9925887775	.9925821608	.0000066167
6.240	.9941683321	.9941618318	.0000065002
6.480	.9954117931	.9954056220	.0000061711
6.720	.9963904631	.9963847508	.0000057123
7.000	.9972719423	.9972668479	.0000050944
7.240	.9978541509	.9978496079	.0000045430
7.480	.9983121846	.9983081827	.0000040018
7.720	.9986724985	.9986690091	.0000034893
8.000	.9989968914	.9989939496	.0000029418
8.240	.9992110775	.9992085561	.0000025214
8.480	.9993795408	.9993773931	.0000021476
8.720	.9995120377	.9995102184	.0000018193
9.000	.9996313064	.9996298162	.0000014901
9.240	.9997100456	.9997087955	.0000012501
9.480	.9997719706	.9997709256	.0000010449
9.720	.9998206712	.9998198008	.0000008704
10.000	.9998645071	.9998638063	.0000007007

so that the region containing the steepest slope is also sampled, and not skipped. On the other hand, choosing an arbitrarily small sampling time induces error because of limited register sizes.

AREAS OF FURTHER INVESTIGATION

The first and foremost matter which should be investigated in the future is determination of the optimum sampling time for a particular problem. Error in the output is a definite function of sampling time. In fact, choosing the wrong sampling time can cause unbounded and oscillatory outputs for an equation which has a bounded monotonic solution. Empirical evidence points toward existence of an optimum sampling time--one which causes minimum error in the output.

It is suggested that an empirical relationship be found, or perhaps a method which first determines the smallest time constant of the equation and utilizes this for determination of the optimum sampling time.

The second area to be investigated concerns extension of this method of solution to differential equations with time-varying coefficients. The method employed in this program is readily adaptable to such equations but it is felt that computation time would increase tremendously. Perhaps a state-space approach is needed here.

SUMMARY

The goal of the introduction has been achieved. The choice of the multiple integrator substitution program, together with exact z-transforms of $(1/s)^n$, resulted in a numerical method of solution whose complexity proved small enough to make an n^{th}-order FORTRAN autoprogram possible.

The output of the autoprogram has moderate accuracy. This was expected, as the method of solution is more primitive than other numerical methods, which, on the other hand, seldom lend themselves to n^{th}-order autoprogramming. If extremely high accuracy is desired, this author suggests that other numerical methods be investigated also. In cases where the initial value of the first derivative is zero, for example, the modified Boxer-Thaler z-form delivers very high accuracy; in some other cases it does not. Thus when dealing with numerical methods, it must be recognized that there is no single method which produces the best solution for every type of problem. With this in mind, the method presented in this thesis is a worth-while addition to those already available.

The autoprogram developed in this thesis represents two basic points of achievement. (1) Elimination of human error in the calculation of coefficients. An example of the possible consequences of errors of this nature is presented by Sidney Lees in reference (6). (2) A completely mechanized high-speed method for obtaining approximate solutions to n^{th}-order differential equations with constant coefficients.

Although the autoprogram was originally designed for equations of order 2 through 15, it has proven itself reliable only up to order 13, inclusively. Limited register sizes in the computer (18 significant figures) cause discrepancies in coefficients for orders higher than 13, and cause unreliable outputs. However, at the time of this writing, the autoprogram has not been tested on computing equipment capable of more than 18 significant figures.

Should any reader be interested in the use of the autoprogram, the author will gladly furnish a deck along with instructions for input data.

ACKNOWLEDGMENTS

The author wishes to thank his major professor, Dr. Charles A. Halijak, for suggesting this research, and for his counsel and assistance in preparing this thesis.

The author is indebted to Mr. David G. Dutra for his assistance and moral support while debugging the FORTRAN autoprogram.

The author expresses his thanks to the Kansas State University Computing Center for providing its facilities.

REFERENCES

1. Halijak, Charles A.
Digital Approximation of Differential Equations by Trapezoidal Convolution, Proceedings of the Central States Simulation Council Meeting on Extrapolation of Analog Computation Methods, Kansas Engineering Experiment Station Special Report No. 10, Vol. 45, No. 7, Kansas State University, Manhattan, Kansas, July, 1961.

2. Halijak, Charles A.
Algebraic Methods in Numerical Analysis of Ordinary Differential Equations, Kansas Engineering Experiment Station, Special Report No. 37, Vol. 47, No. 9, Kansas State University, Manhattan, Kansas, September, 1963.

3. Criswell, M. L.
An Algorithm for Computing the z-transform of $t^n/n!$, Kansas Engineering Experiment Station, Special Report No. 37, Vol. 47, No. 9, Kansas State University, Manhattan, Kansas, September, 1963.

4. Kreyszig, E.
Advanced Engineering Mathematics, John Wiley & Sons, Inc., New York, New York, 1962; p. 621.

5. Naumov, B.
Approximate Method for Calculating the Time Response in Linear, Time-varying, and Nonlinear Automatic Control Systems, Transactions ASME, Journal of Basic Engineering, Vol. 83, Series D, No. 1, pp. 109-118, March, 1961.

6. Lees, S.
z-Form Technique, Proceedings of the IEEE, Vol. 53, No. 11, November, 1965, p. 1777.

APPENDIX

LISTING OF THE AUTOPROGRAM
FORTRAN 4 LANGUAGE
SOURCE PROGRAM

```
C THE PICKER-PROGRAM FOR ORDINARY DIFF. EQUATIONS                          *
C THIS PROGRAM WILL ACCEPPT ANY ORDINARY DIFF. EQUATION WITH CONSTANT       *
C COEFFICIENTS, AND OF ANY ORDER BETWEEN 2 AND 13.                         *
C IN THIS FORM THE PROGRAM IS GEARED TO THE IBM 1410-PR-155                *
      INTEGER A,R,U,SUM,DENOM,BICOEF,A3,A1,A2
      DIMENSION A(105),A1(16),A12T(16),A3(14,16),AO1(16),Y(15),A3T(16)
      COMMON N,XO,T,A2(16),NPI,AO2T(16),YOD(15),YO,C(16),CON
   95 FORMAT(I3,E24.18,2F10.5)
   96 FORMAT(E24.18)
  100 FORMAT(4I20)
*  105 FORMAT(1HL,76H*** BE SURE TO DOUBLE-CHECK YOUR FORCING FUNCTION IN
     1 THE W(K)-SUBROUTINE ***)
  106 FORMAT(1HL,38HLISTING OF THE C(I) IN ASCENDING ORDER)
  107 FORMAT(1HL,56HLISTING OF Y-ZERO AND THE Y-ZERO-DOTS IN ASCENDING O
     1RDER)
*  108 FORMAT(1H1,32HYOU READ-IN THE FOLLOWING VALUES,/1HL,1X,1HN,20X,4HX
     1NOT,6X,4HTIME,9X,1HT,/1HK)
*  203 FORMAT(1H1,47X,38HLISTING OF THE ITERATED VALUES OF Y(K),/1HL,4(5X
     1,4HTIME,20X,4HY(K)))
*  204 FORMAT(1HK,4(F9.4,1X,E23.17)/1HK,4(F9.4,1X,E23.17)/1HK,4(F9.4,1X,E
     123.17)/1HK,3(F9.4,1X,E23.17))
  206 FORMAT(1HK,6E20.10/1HK,20X,5E20.10/1HK,20X,5E20.10)
      READ(1,100) A
    1 DO 3 I=1,14
      DO 3 J=1,14
    3 A3(I,J)=0
      DO 4 I=1,16
      A2(I)=0
      AO1(I)=0.0
      YOD(I)=0.0
    4 AO2T(I)=0.0
    5 READ(1,95) N,XNOT,TIME,T
      WRITE(3,108)
      NI=N-1
      NH=N/2
      NPI=N+1
      XO=.5*XNOT
      WRITE(3,95) N,XNOT,TIME,T
      READ(1,96) (C(I),I=1,NPI)
      READ(1,96) YO,(YOD(I),I=1,NI)
      WRITE(3,106)
      WRITE(3,206)(C(I),I=1,NPI)
      WRITE(3,107)
      WRITE(3,206)YO,(YOD(I),I=1,NI)
      WRITE(3,105)
      IF(T.EQ.-2.*C(1)/C(2)) T=T*.5
      NF=N
```

```
      L=IFAC(NF)
      DO 111 K=1,NPI
      M=K+1
  111 A1(K)=L*(-1)**M/(IFAC(K-1)*IFAC(N+1-K))
      L2=IFAC(NI)
      DO 112 K=1,N
      M2=K+1
  112 A2(K)=L2*(-1)**M2/(IFAC(K-1)*IFAC(N-K))
      DO 113 K=1,N
      J=N+2-K
  113 A2(J)=A2(J)+A2(J-1)
      DO 130 K=1,NI
      DO 130 R=1,NH
      SUM=0
      DO 110 U=1,K
      DENOM=IFAC(R-U)*IFAC(NI+U-K-R)
      IF(DENOM.EQ.0) GO TO 110
      BICOEF=IFAC(NI-K)/DENOM
      IVEC=(K*(K-1))/2+U
      SUM=SUM+A(IVEC)*BICOEF*(-1)**(R-U)
  110 CONTINUE
  120 A3(K,R)=SUM
      L=N-R
      NK1=N-K+1
  130 A3(K,L)=SUM*(-1)**NK1
      DO 134 I=1,NI
      A3(I,NPI)=0
      DO 133 J=1,NI
      K=NPI-J
  133 A3(I,K)=A3(I,K-1)
  134 A3(I,1)=0
      DO 30 J=1,NPI
      A12T(J)=C(1)*FLOAT(A1(J))+(C(2)*.5*T)*FLOAT(A2(J))
      A3T(J)=0.
      I=1
      TP=T
      GO TO 29
   28 I=I+1
      IF(I.GT.NI) GO TO 30
   29 IP2=I+2
      TP=TP*T
      A3T(J)=A3T(J)+(C(IP2)*FLOAT(A3(I,J))/FAC(I))*TP
      GO TO 28
   30 A12T(J)=A12T(J)+A3T(J)
      A2(1)=0
      DO 132 J=2,N
      K=NI*(NI-1)/2+J-1
  132 A2(J)=A(K)
      A2(NPI)=0
      IF(YO.EQ.0.) GO TO 136
      L11=IFAC(NI)
      DO 230 K=1,N
      M11=K+1
```

```
230 A1(K)=L11*(-1)**M11/(IFAC(K-1)*IFAC(N-K))
    A1(NPI)=0
    D=2.*C(1)+C(2)*T
    DO 231 J=1,N
231 AO1(J)=(FLOAT(A1(J))*D+A3T(J))*YO*.5
    GO TO 33
136 DO 32 I=1,NI
    IF(YOD(I).NE.0.) GO TO 33
 32 CONTINUE
    GO TO 237
 33 DO 234 J=1,N
    AO2T(J)=0.
    TP=T
    K=1
    GO TO 233
232 K=K+1
    IF(K.EQ.N) GO TO 234
    TP=TP*T
233 KPI=K+1
    AO2T(J)=AO2T(J)+(CO(KPI)*FLOAT(A3(K,J))/FAC(K))*TP
    GO TO 232
234 CONTINUE
235 DO 236 J=1,N
236 AO2T(J)=AO2T(J)+AO1(J)
237 WRITE(3,203)
    TI=TIME/(T*FLOAT(N))
    IT=IFIX(TI)+1
    CON =T**N/FAC(NI)
    DO 49 I=1,IT
    DO 48 K=1,N
    KP=(I-1)*N+K
    IF (I.GT.1) GO TO 40
    GO TO 41
 40 Y(K)=Y(K)*(-A12T(NPI))+W(KP)
    GO TO 42
 41 Y(K)=W(KP)
 42 IF (KP.EQ.1) GO TO 48
    IOBJ=KP
    IF(I.GT.1)IOBJ=N
    DO 47 J=2,IOBJ
 44 KL=K+1-J
    IF (KL.LE.0) GO TO 45
    GO TO 46
 45 KLN=N+KL
    Y(K)=Y(K)-A12T(J)*Y(KLN)
    GO TO 47
 46 Y(K)=Y(K)-A12T(J)*Y(KL)
 47 CONTINUE
 48 Y(K)=Y(K)/A12T(1)
    DO 485 K=1,N
485 AO1(K)=FLOAT((I-1)*N-1+K)*T
 49 WRITE(3,204)(AO1(K),Y(K),K=1,N)
    END
```

LISTING OF THE FUNCTION SUBROUTINES

FACTORIAL SUBROUTINES

```
      FUNCTION IFAC(N)
      IF(N)10,20,30
10    IFAC=0
      RETURN
20    IFAC=1
      RETURN
30    IFAC=1
      NRAN=N
      DO 40 I=1,NRAN
40    IFAC=IFAC*I
      RETURN
      END
```

```
      FUNCTION FAC(N)
      IF(N) 50,60,70
50    FAC=0.
      RETURN
60    FAC=1.
      RETURN
70    FAC=1.
      NRAN=N
      DO 80 I=1,NRAN
80    FAC=FAC*FLOAT(I)
      RETURN
      END
```

THE CO(K) SUBROUTINE

```
      FUNCTION CO(K)
      INTEGER A2
      COMMON N,XO,T,A2(16),NPI,AO2T(16),YOD(15),YO,C(16),CON
      CO=0.
      DO 90 I=1,K
      KI=K-I
      IF(KI.EQ.0)CO=CO+C(I)*YO
      IF(KI.EQ.0) GO TO 90
      CO=CO+C(I)*YOD(KI)
   90 CONTINUE
      RETURN
      END
```

THE W(K) SUBROUTINE

```
      FUNCTION W(K)
      INTEGER A2
      COMMON N,XO,T,A2(16),NPI,AO2T(16),YOD(15),YO,C(16),CON
      IF(K.GT.N) GO TO 81
      W=XO*FLOAT(A2(K))*CON+AO2T(K)
      IF (K-1) 84,84,82
   81 W=0.
   82 DO 83 I=1,N
      KI=K-I
      IF (KI.EQ.0) GO TO 84
      TT=KI
      TVAR=TT*T
C THE NEXT CARD IS YOUR FORCING FUNCTION, THE TIME VARIABLE
      MUST BE TVAR
      X= A FUNCTION OF TVAR
   83 W=W+FLOAT(A2(I))*X*CON
   84 RETURN
      END
```

A FORTRAN AUTOPROGRAM FOR SOLVING ORDINARY LINEAR DIFFERENTIAL EQUATIONS WITH CONSTANT COEFFICIENTS USING EXACT Z-TRANSFORMS OF $(1/s)^n$ AND TRAPEZOIDAL CONVOLUTION

by

WILLIAM A. PICKER

B. S., Kansas State University, 1964

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1966

Numerical transform techniques were introduced into electrical engineering by Tustin in 1947. Since then, several investigators have produced various methods for digital solution of ordinary linear and nonlinear differential equations. It has been shown by Halijak that all of these methods have trapezoidal convolution as their fundamental basis.

This thesis uses the multiple integrator substitution program, together with exact z-transforms of $(1/s)^n$, for solving ordinary differential equations with constant coefficients. This method, like all other numerical methods, requires lengthy algebraic calculations for determining coefficients of the recurrence relation. In order to eliminate manual algebraic calculations, and associated human error, a FORTRAN program is developed which determines the recurrence relation, and iterates this recurrence relation to obtain the discrete solution of the differential equation.

This program is an "autoprogram", since it accepts any n^{th}-order ($2 \leq n \leq 13$) differential equation with constant coefficients without additional programming effort.

The mathematical derivation of the n^{th}-order discrete solution, both with and without existing initial conditions, is included. Development of the autoprogram is demonstrated with flow charts, and a listing of the autoprogram is given in the appendix.

The result is a completely mechanized, high-speed method for solving an n^{th}-order differential equation with constant coefficients, resulting in moderate accuracy solutions.